Literate Style Co-programming with GPT-4 as a Copilot - An Experiment and Study

Grzegorz Wierzowiecki

2023-05-19

Abstract

This study explores literate programming with GPT-4 as a copilot, emphasizing enhanced workflow design and autonomous agent development. The research elucidates the value of literate programming in fostering structured problem-solving and improved communication about coding. It identifies GPT-4's strengths and weaknesses, highlighting its ability to maintain intent consistency and recover from errors with appropriate cues. Future research directions include improving AI agent efficiency, enhancing human comprehension, and addressing the semantic meaning of programs. (abstract by GPT4, article by human)

Intro

This is experiment of making and debugging script with GPT4.

Goal was to understand better it's weaknesses and failure modes and how to recover.

I believe that such understanding helps in designing better workflows, both, for copilot assistant way of operating, and for designing autonomous agents.

It is educational example of conversation with GPT4 used to define readme of script, code, debug it and fix.

I attach script as educational resource, to serve as inspiration.

Literate Style of programming

Since beginning I had gut feeling that Literate Programming style, maybe great match with LLM.

However after reading about studies on effectiveness of CoT (chain-of-thought) Prompting (Chain-of-Thought Prompting Elicits Reasoning in Large Language Models https://arxiv.org/abs/2201.11903), I realised that there maybe more to it, as to me it's basically like another flavour of chain-of-thought.

Therefore, I encouraged GPT-4 to use technique of https://en.wikipedia.org/wiki/Literate_programming by Donald Knuth.

I believe that old idea of Literate Programming from that time is great fit when working with LLM assistant, as it both, helps to establish meaning and intent of code for mutual sides, as well as for user to learn during the process, not only about code, but how to describe in structured way thinking, using human language, that I believe may improve mutual communication between humans, about complex structured matters, like computer code.

Observations

- After dialog and encouraging it to ask me questions to make sure we end up with good requirements (which was very good, as revealed its misunderstandings of some of my intents), I encouraged to start by making README.md, basically to be requirements document and specification. This served purposes: to have nice README.md, to verify understanding of whole project and work it is planning to do, having reference point to refer to and eventually copy&paste, in case long conversation would need reminding of what is to be done.
- It was keeping consistency of intent very well
  - probably due to large volume of text capturing ideas and semantics
- However, when fixing or rewriting routines, it was sometimes hallucinating about how they looked earlier, nor important bits, like e.g. how POST URL for request should look like etc. However, after pointing out minor details, and encouraging it to insert more debugging information giving opportunity to reason from program outputs, it was able to recover successfully and finish task.
  - Missing important program bits, was

probably due to large volumes of text, so it also lost critical bits. However overall, I felt that it kept semantic consistency of program over long conversation way better then when working in long conversations with not commented code.

To investigate in the future:

- especially when designing Agents I will consider:
- start with interview to produce specification document
- then make step to design high level execution plan
- and try to push it to break program into smaller functions.
- and implement smaller function one at a time.

I have impression, that it often operates on granularity of whole function when applying fixes. That's why my intuition and curiosity if it performs better when working with smaller functions, therefore if cross functional issue would appear to have enough tokens to comprehend all required pieces at once, that may :

- require "fixing scheduler/assistant" agent - that would chose relevant parts/functions for given issue, so "fixer agent" would be provided with fresh memory of pieces that are most relevant to issue.

More future work:

- As you study dialog, some may argue that there is room for improvement in regards to Literate Programming style, principles, benefits, techniques, therefore:
  - More Literate Programming Prompt Engineering work maybe required
  - Studying what principles of Literate Programming make style helpful for humans, and which parts make it more helpful for LLMs, like Chain-of-Thought Prompting.
  - For what kinds of tasks those techniques are giving benefits, and what benefits.
  - Finally identify, which parts to focus on when goal is to increase efficiency of AI Agents performing tasks, and on which if goal is to optimize for future human comprehension.

Author's guess:

- It depends on program, task, topic.
- If topic semantics is close to algorithmic, then program structures maybe sufficiently carrying enough semantic information, themselves to reason.
- However if program is to address some domain space from ontologies outside of computer program realm, than carrying semantics of logic requires extra means, therefore comments, like Literate Programming may help to maintain and keep program more resilient long term to semantic transitions mismatch, more robust, addressing better actual purpose, and giving possibility for future iterations, with additional tasks like "generate more tests", "are all needs addressed", "please interview me to ensure if all needs are covered", etc. Therefore focusing on actual semantic meaning of program in first place, and having code as consequence. I expect also such refactoring providing sometimes, completely new code solutions, that solve same problem and maybe even better, because starting point is not a code but problem statement, and resining. Finally reasoning itself, captured in literature programming can be investigated and refined, and analysed. Therefore, program maybe correct, but there maybe flaws in reasoning that lead to this program. Having reasoning captured allows to investigate reasoning in first place.

Some notes

This document is to capture dialog to study, and my observations and conclusions. (Please consider that some of prompts required reediting, based on responses, due to linear nature of document, capturing those attempts, to illustrate prompt engineering, fine tuning aspects, would probably more decrease readability then provide extra value, so I stick to final format of conversation).

If you see potential to improve this article, feel free to shoot PR (pull request) :).

Thank you for interest in reading this writeup.

References

- (chain-of-thought) Prompting (Chain-of-Thought Prompting Elicits Reasoning in Large Language Models https://arxiv.org/abs/2201.11903)
- https://en.wikipedia.org/wiki/Literate_programming by Donald Knuth.